In [1]:
```
## import library
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import quandl
import numpy as np
import matplotlib.pyplot as plt  #for plotting
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn import preprocessing,cross_validation
from sklearn.svm import SVR
from mlxtend.regressor import StackingRegressor
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor,  GradientBoostingRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import RobustScaler
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
```

```
/root/anaconda2/lib/python2.7/site-packages/sklearn/cross_validation.py:41: DeprecationWarning:
This module was deprecated in version 0.18 in favor of the model_selection module into which all
the refactored classes and functions are moved. Also note that the interface of the new CV itera
tors are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
## data source
quandl.ApiConfig.api_key = "zuiQMfguw3rRgLvkCzxk"
df=quandl.get('WIKI/GOOGl')
```

In [3]:
```
##data summary
df.head()
```

Out[3]:

| | Open | High | Low | Close | Volume | Ex-Dividend | Split Ratio | Adj. Open | Adj. High | Adj. Low | Adj. Close |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Date** | | | | | | | | | | | |
| **2004-08-19** | 100.01 | 104.06 | 95.96 | 100.335 | 44659000.0 | 0.0 | 1.0 | 50.159839 | 52.191109 | 48.128568 | 50.322842 |
| **2004-08-20** | 101.01 | 109.08 | 100.50 | 108.310 | 22834300.0 | 0.0 | 1.0 | 50.661387 | 54.708881 | 50.405597 | 54.322689 |
| **2004-08-23** | 110.76 | 113.48 | 109.05 | 109.400 | 18256100.0 | 0.0 | 1.0 | 55.551482 | 56.915693 | 54.693835 | 54.869377 |
| **2004-08-24** | 111.24 | 111.60 | 103.57 | 104.870 | 15247300.0 | 0.0 | 1.0 | 55.792225 | 55.972783 | 51.945350 | 52.597363 |
| **2004-08-25** | 104.76 | 108.00 | 103.88 | 106.000 | 9188600.0 | 0.0 | 1.0 | 52.542193 | 54.167209 | 52.100830 | 53.164113 |

In [4]:
```
## redefining data adding removin feture
### create the specfic ammount of label and feture
df1=df[['Adj. Open','Adj. High','Adj. Low','Adj. Close','Adj. Volume']]

###redefining the data
### adding some feture to the datasets
df1['volatility']=(df1['Adj. High']-df1['Adj. Close'])/df1['Adj. Close']
df1['PCT_Change']=(df1['Adj. Close']-df1['Adj. Open'])/df1['Adj. Open']
```

In [5]:
```
## making final dataframe
df1=df1[['Adj. Close','volatility','PCT_Change','Adj. Open','Adj. Volume']]
```

In [6]:
```
## setting the target column
forcast_col='Adj. Close'
```

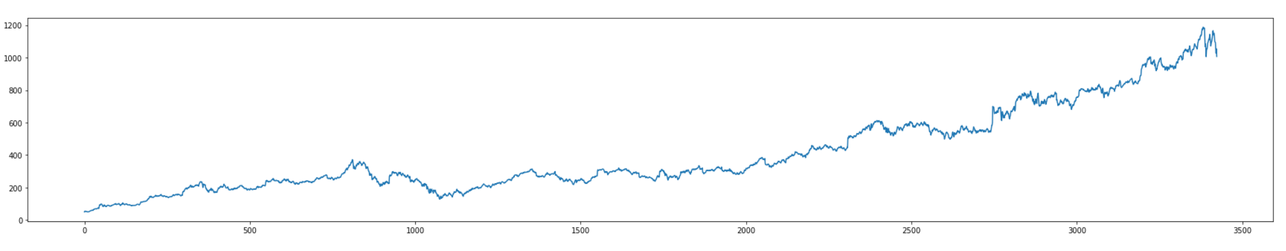
In [7]:
```
## deal with the null data
df1.fillna(-999999,inplace=True)
```

In [8]:
```
## for predicting one percent of the data
import math
forcast_out = int(math.ceil(.1*(len(df1))))
print forcast_out
```

```
343
```

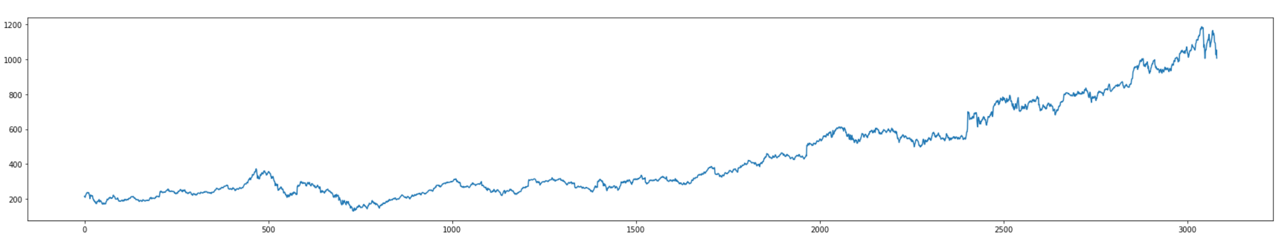
In [9]:
```
## displaying the previous output
Y=df1[forcast_col]
X=range(len(df1[forcast_col]))
fig_size=[30,5]
plt.rcParams["figure.figsize"] = fig_size
plt.plot(X,Y)
```

Out[9]: [<matplotlib.lines.Line2D at 0x7f158b090990>]




In [10]:
```
##storing the previous data in  a dataframe
df1['label'] = df[forcast_col].shift(-forcast_out)
y1 = df1['label']
x1=range(len(df1['label']))
fig_size=[30,5]
plt.rcParams["figure.figsize"] = fig_size
plt.plot(x1,y1)
```

Out[10]: [<matplotlib.lines.Line2D at 0x7f158b182150>]




In [11]:
```
## dropping the first column which is the output
X=np.array(df1.drop(['label'],1))
```

In [12]:
```
##scale the data
X=preprocessing.scale(X)
X=X[:-forcast_out]   ##data what is known
X_lately=X[-forcast_out:] ##data we predict
df1.dropna(inplace=True)
```

In [13]:
```
Y=np.array(df1['label'])
```

In [14]:
```
##split the training and testing data
xtrain,xtest,ytrain,ytest=cross_validation.train_test_split(X,Y,test_size=0.2)
```

In [62]:
```
## training separtely the classifier
##first knn
n_neighbors=1
clf1 = KNeighborsRegressor(n_neighbors)  # create a classifire object
clf1.fit(xtrain,ytrain) # train data related with fir() method
accuracy1=clf1.score(xtest,ytest) # test data related with score() method
print "the accuracy is "+str(accuracy1)
```

```
the accuracy is 0.8918664367582111
```

In [55]:
```
## second linear regression
from sklearn.linear_model import LinearRegression
clf2 = LinearRegression()  # create a classifire object
clf2.fit(xtrain,ytrain) # train data related with fir() method
accuracy2=clf2.score(xtest,ytest) # test data related with score() method
print "the accuracy is "+str(accuracy2)
```

```
the accuracy is 0.8797715162334666
```

In [17]:
```
## third support vector machine
from sklearn import svm
clf3 = svm.SVR()  # create a classifire object
clf3.fit(xtrain,ytrain) # train data related with fir() method
accuracy3=clf3.score(xtest,ytest) # test data related with score() method
print "the accuracy is "+str(accuracy3)
```

```
the accuracy is 0.743281531867497
```

In [49]:
```
clf4 = RandomForestRegressor(max_depth=2, random_state=0,n_estimators=100)
clf4.fit(xtrain,ytrain) # train data related with fir() method
accuracy4=clf4.score(xtest,ytest) # test data related with score() method
print "the accuracy is "+str(accuracy4)
```

```
the accuracy is 0.8867107516823005
```

## applying the stacking method we developed

In [19]:
```
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models

    # we define clones of the original models to fit the data in
    def fit(self, X, y):
        self.models_ = [clone(x) for x in self.models]

        # Train cloned base models
        for model in self.models_:
            model.fit(X, y)

        return self

    #Now we do the predictions for cloned models and average them
    def predict(self, X):
        predictions = np.column_stack([
            model.predict(X) for model in self.models_
        ])
        return np.mean(predictions, axis=1)
```

In [20]:
```
averaged_models = AveragingModels(models = (clf1, clf2, clf3, clf4))
```

In [21]:
```
averaged_models.fit(xtrain,ytrain)
```

Out[21]:
```
AveragingModels(models=(KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=2, p=2,
          weights='uniform'), LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normal
ize=False), SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsi..._estimators=100, n_jobs=1,
           oob_score=False, random_state=0, verbose=0, warm_start=False)))
```

In [22]:
```
accuracy=averaged_models.score(xtest,ytest)
```

In [23]:
```
accuracy
```

Out[23]: 0.9072652889535424

## This is better than the individual one

In [24]:

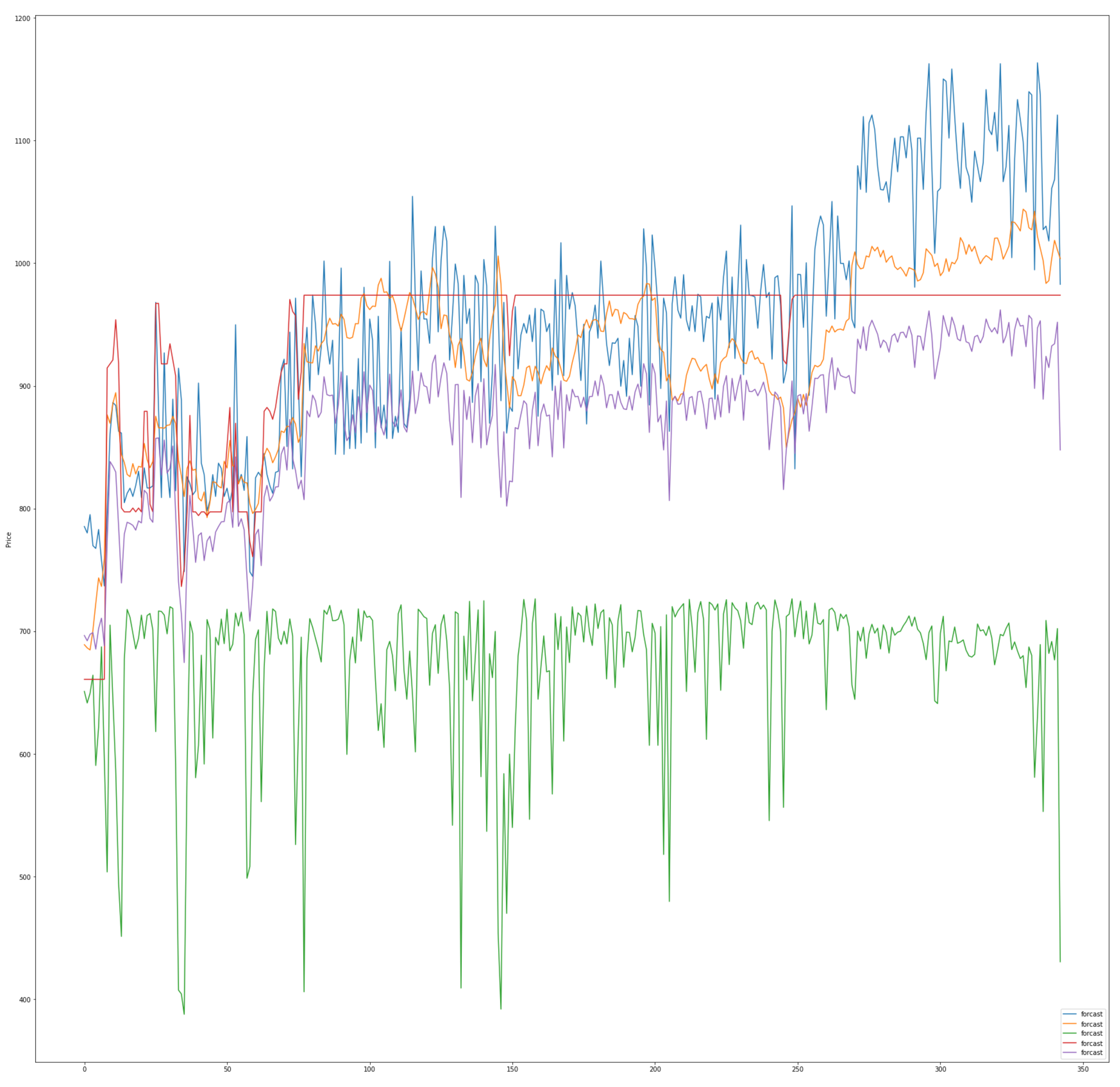
In [37]:
```
df2=pd.DataFrame()
df3=pd.DataFrame()
df4=pd.DataFrame()
df5=pd.DataFrame()
df6=pd.DataFrame()
```

In [38]:
```
forcast_set1=clf1.predict(X_lately)
forcast_set2=clf2.predict(X_lately)
forcast_set3=clf3.predict(X_lately)
forcast_set4=clf4.predict(X_lately)
final_forcast_set=averaged_models.predict(X_lately)
df2['forcast']=np.array(forcast_set1)
df3['forcast']=np.array(forcast_set2)
df4['forcast']=np.array(forcast_set3)
df5['forcast']=np.array(forcast_set4)
df6['forcast']=np.array(final_forcast_set)
```

In [52]:
```
fig_size=[30,30]
plt.rcParams["figure.figsize"] = fig_size
df2['forcast'].plot()
df3['forcast'].plot()
df4['forcast'].plot()
df5['forcast'].plot()
df6['forcast'].plot()
plt.legend(loc=4)

plt.ylabel('Price')
```

Out[52]: Text(0,0.5,'Price')




## The Orange one is our improved stacked model output, less noise more accuracy